TFS and Git: a comparison

If you want an integrated, enforced process, use TFS. If you value developer freedom and individual work strategies, use git.

The players:

[**TFS**](http://msdn.microsoft.com/en-US/vstudio/ff637362.aspx) (Team Foundation Server) is Microsoft's answer to version control. TFS was designed big and bad-ass enough to support development of Visual Studio, like 5 million lines of code. [1] Since not everyone lives in Redmond, TFS tries to minimize network traffic and keep the work on the server.

[**git**](http://git-scm.com/) is open-source. It was created to manage the source code of the Linux kernel, roughly 15 million lines of code. [2] Development is distributed among thousands of contributors around the world. Many features and versions are in process all the time.

Why should your team use one or the other?

First, if you don't use Visual Studio, forget about TFS. [Correction: it can be done.]

Second, do you want version control or a full suite including a build server, work item tracking, and analytics? git is version control only. Github has some of the same bells and whistles as TFS, but Github is outside the scope of this post. [update: TFS has started to embrace git, and you can use git as version control within TFS.]

Here's a high-level comparison of features, for those who enjoy such formats. Below that, find discussion of the important feature differences, user experiences, and the one question that will tell you which is right for you.

| **Feature** | **TFS** | **git** |
| --- | --- | --- |
| Save source code | Yes | Yes |
| Retain all version history | Yes | Yes |
| Group changes into sets | Yes (file level only) | Yes |
| Automatic change detection | in 2012 | Yes |
| Branch | Yes (not easy) | Yes (very easy) |
| Merge between related branches | Yes | Yes (very easy) |
| Safe merge between unrelated branches | No | Yes |
| Offline access to history | No | Yes |
| Offline commit | No | Yes |
| Offline commit | No | Yes |
| Sneaky automatic merges on the server | Yes | No |
| Enforce requirements before commit | Yes | No |
| Private local branches | No | Yes |
| Learning curve for users | Low | Medium |
| Learning curve for administrators | Very high | Medium-high |
| GUI support | Strong | Poor |
| Conditional commit (if tests work) | Yes | No |
| Save IDE state | Yes | No |
| Work item tracking | Yes | No |
| Automated builds and tests | Yes | No |
| Analytics and charting | Yes | No |
| Manual test tracking | Yes | No |
| Installation | 1/2 day | 10 minutes |
| Deployment, central and local | days to months | 1 day |
| Free online hosting | Yes | Yes |
| Cost | $$$ | free |
| Security | Yes | No |

What does this chart say? It says, "**Git does version control, and does it well**" and "**TFS does version control and build automation and work item tracking and test tracking and analytics in a SQL Server data cube with graphs and charts."**

TFS is a whole development workflow suite. It integrates all its parts, and then integrates with Visual Studio, Sharepoint, and Active Directory. Git doesn't impose or supply any of these parts; there are other solutions both open-source and commercial. If you want to select each item of furniture separately, hunting through different stores until you find the best chair and couch and table for you, then git is your source control rug. If you prefer to stop at Ethan Allen and choose a coordinating set, that's TFS.

Michael's company uses TFS for source control. On work item tracking, he says, "We haven't made use of this feature because we haven't adapted our business process to it... which come to think of it doesn't make sense." You don't have to use all the features of TFS.

Choosing to use all of TFS touches the whole business of producing software. An impact like this can lead to a lot of meetings, and that is why J.R. Roy and [James](https://twitter.com/JamesKovacs) both said deployment can take months, even though installation is fast.

I'll highlight a few killer features of TFS, and then a few of git.

If you like tight, granular control over security, TFS has this. Use AD groups or define your own; assign read and write permissions down to the individual file level if needed. In contrast, access to  whole git repositories is enforced by the file system.

The tight integration of TFS components enforces traceability: for instance, organizations can require that each changeset be associated with a work item. Luc says TFS is "good stuff. All code is related to a work item, everything is linked together."

TFS has an explicit checkout process. This is both bad and good. In TFS 2010 and earlier, files are read-only until Visual Studio checks them out, which drives people nuts. External editors don't integrate. In TFS 2012 there's an improvement: changes are noticed automatically, and the checkout performed behind the scenes. This explicit checkout process is good because when the server knows who is working on each file, a developer can find out who else is changing the file they're about to slaughter. I've wished for this in IntelliJ IDEA with git.

Then again, as Maxime points out, "If you're working on the kernel of Linux you don't want to know" whether sixty other developers in fifty different feature branches are working on the same file.

There's another feature of TFS that will either make you say "fantastic!" or "run away!" TFS requires SQL Server, and it uses the database to store all kinds of tracking data about work items, commits, code quality, and build history. It outputs an entire data cube to generate burndown charts, bugs over time, and all those high-level pictures that give managers the illusion that they know what's going on in the development team.

Now for git's killer features. Git is a distributed version control system (DVCS), which means that every developer's copy can access every version of every file ever, even on an airplane. Incidentally, this serves as a free offline backup for every dev who takes their laptop home. More importantly, people can view history, save their own commits, and branch all without a connection to the central repository. The central depot is accessed only when explicitly requested.

There's a subtler advantage to having a whole repository locally: saving work is separated from sharing work. Developers can save a series of changes and then test thoroughly before transferring those changes to the central depot and build server. As Tri put it: "I like being able to save my code without making my friends suffer."

Then again, as [Etienne Tremblay](https://twitter.com/tegaaa) sees it, the extra step of saving locally and then sharing "adds a layer of complexity." It is another step the developer has to think about, another decision to be made.

Good news for the developer who wants the privacy of git while his organization prefers TFS: integration. There's [git tfs](https://github.com/git-tfs/git-tfs), which is a git plugin, and [git-tf](http://gittf.codeplex.com/), written by Microsoft. Git repositories and TFS can talk to each other. Try git without impacting your team.

At this point, git fans are screaming, "What about branching and merging??" The subtleties of the differences here are beyond the scope of this post, but in brief: branches and merges are orders of magnitude smaller and simpler in a git workflow. I create git branches and merge changes several times a day. Contrast with Daniel, a TFS user: "We tried branching one time, but we didn't end up using it."

Then again, Etienne points out, "dealing with branches is very complicated for people." Maybe developers don't want this alternative.

At its core, TFS is centralized. To reduce network traffic, Microsoft "chose a design where the vast majority of information (including information about your local workspace) was stored on the server." [1] Git, on the other hand, keeps every local copy fully independent. Even the central depot is only central by convention. The happy rumor is that TFS is embracing more of the distributed model in future releases.

Now for the real, defining difference between TFS and git. TFS says, "This is how to run your process. Follow these steps." There are customization points; these are a lot of work and will be decreed by the organization. Git says, "it's your repository, do what you need to do. Then choose how to tell the story of your project." Git gives you, as a developer and as a team, a lot of choices. That's not always a good thing.

[1] http://blogs.msdn.com/b/bharry/archive/2011/08/02/version-control-model-enhancements-in-tfs-11.aspx  
[2] <http://en.wikipedia.org/wiki/Linux_kernel>